Project Report 2:
Rutvij Dhotey

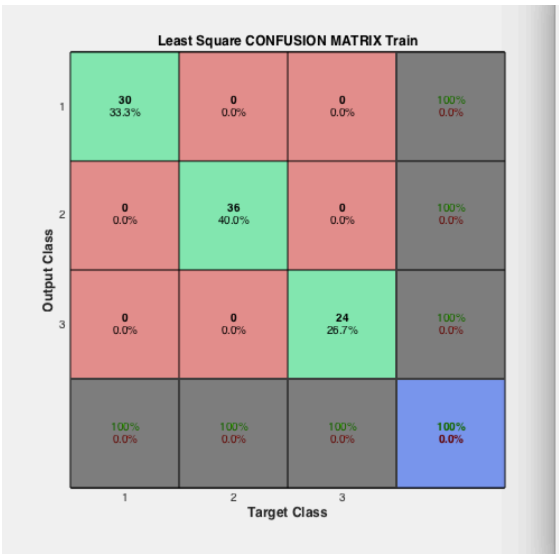
Pattern Recognition and Machine Learning

Part 1: Classification using Least Square Sum

Least Square Sum Classification is a supervised learning method, thus we provide data before we test.

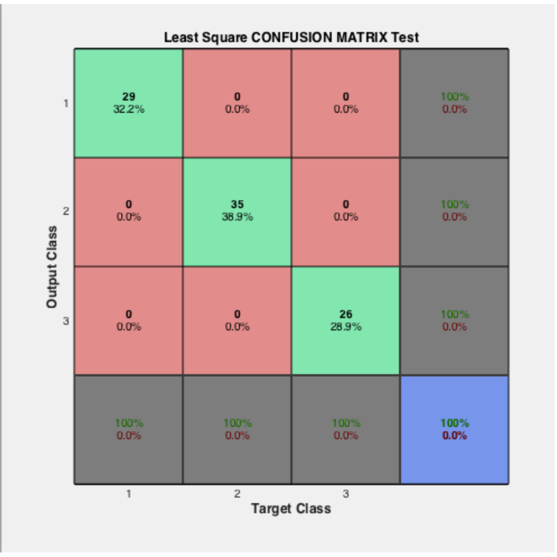
1. We first generate the training data points and Training Outputs by using the Wine Data provided. These 90 points are 13dimensional. So out training set is a 90x13 data matrix. We add a column of ones for the 0th order .
2. Initially we solve the problem of linear regression by the technique of error minimisation. The minimising of sum of squares error function will give us a straight forward vector formula as calculated in the first homework.
   The Formula can be stated as: X is the training input and T is the training output.
   $$\mathbf{W^{*}= (X^{T}X)^{-1} * X^{T} * T}$$
3. Thus here W* is the coefficient parameters that we calculate to train wrt. the input / training data.
4. As we have to calculate the output y(X,W*)
   $$\mathbf{y(X,W^{*})= \sum_{i=0}^{M} W0 + W1 * Xi + W2 * Xi^{2} \ldots\ldots..WM * Xi^{\wedge}M}$$
5. After the calculation of the Y output matrix, we can then use the threshold technique or the max column method to classify the samples. The column having the maximum value will be the class of that specific data sample.
6. We then make a confusion matrix and show the percentage of classification.

Observations:



Least Square CONFUSION MATRIX Train

Output Class \ Target Class	1	2	3	
1	30 33.3%	0 0.0%	0 0.0%	100% 0.0%
2	0 0.0%	36 40.0%	0 0.0%	100% 0.0%
3	0 0.0%	0 0.0%	24 26.7%	100% 0.0%
	100% 0.0%	100% 0.0%	100% 0.0%	100% 0.0%

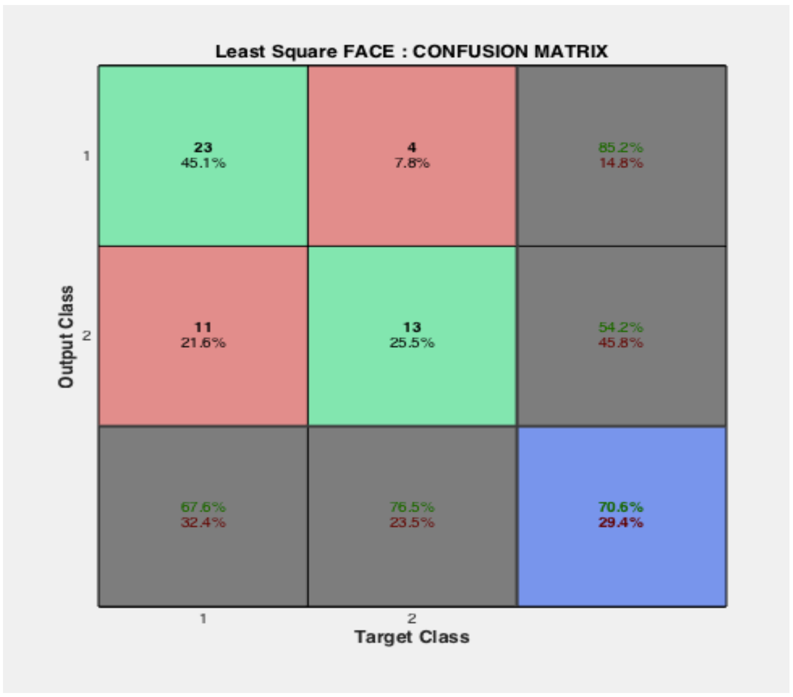


Least Square CONFUSION MATRIX Test

Output Class \ Target Class	1	2	3	
1	29 32.2%	0 0.0%	0 0.0%	100% 0.0%
2	0 0.0%	35 38.9%	0 0.0%	100% 0.0%
3	0 0.0%	0 0.0%	26 28.9%	100% 0.0%
	100% 0.0%	100% 0.0%	100% 0.0%	100% 0.0%

As we can clearly see, we get a 100% accuracy of classifying all the data points into the respective classes. The Observation that is important here is that the more the dimensions, we get a better understanding and more data for training the training vector(W). Thus as we have 13 dimensions here, we see that the data is better classified. Also the number of data points used for training(90) are enough to provide us with perfect classification. But Least squares is not the most effective way to classify. Because we consider that the LSE is used under the assumption that the data is having a Gaussian Distribution (Continuous data), we can say that it doesn't work well with discrete data. We can see this effect on the FACE dataset . We do not get a 100% accuracy. Rather it is much less.

Observations for Face Data:



Overfitting:

As we have seen in the previous project, the problem of overfitting arises when we have many different dimensions(parameters) based on which we need to classify the data. It also depends on the amount of data points available. In this , we cannot observe any overfitting(only for LSE) as both training and testing data is completely classified. However sometimes depending on the data, the confusion matrix might change.

Part 2: Fisher Linear Discriminant Analysis

The Fisher Linear discriminant is used to project a high dimensional data onto a C-1 dimensional space where in C is the number of classes used. Thus we reduce the high dimensionality by maximising the ratio of Inter Class Mean and minimising the Intra Class Variance. This can be stated as a function of a a variable W as,

$$J(\mathbf{w}) = \frac{\mathbf{w}^{\mathrm{T}}\mathbf{S}_{\mathrm{B}}\mathbf{w}}{\mathbf{w}^{\mathrm{T}}\mathbf{S}_{\mathrm{W}}\mathbf{w}}$$

In the above equation ,

$$\mathbf{S}_{\mathrm{B}} = \sum_{k=1}^{K} N_k(\mathbf{m}_k - \mathbf{m})(\mathbf{m}_k - \mathbf{m})^{\mathrm{T}}$$

This is the inter-class covariance/ Total Class Covariance before projection.

m_k is the mean with respect to each dimension, where k is the class. So in our case we have 13 dimensions and 3 classes. So we will have m_1 m_2 m_3 (used as u1 u2 u3 in code).

We also need to calculate S_w
But in order to do that we calculate :

$$\begin{aligned}\mathbf{S}_k &= \sum_{n \in \mathcal{C}_k} (\mathbf{x}_n - \mathbf{m}_k)(\mathbf{x}_n - \mathbf{m}_k)^{\mathrm{T}} \\ \mathbf{m}_k &= \frac{1}{N_k} \sum_{n \in \mathcal{C}_k} \mathbf{x}_n\end{aligned}$$

Where in S_k is the intra-class Variance for each class.

And hence Sw,

$$\mathbf{S}_{\mathrm{W}} = \sum_{k=1}^{K} \mathbf{S}_k$$

Sw is the total Intra-Class Variance. Where in K is the number of classes.

After we find the two Sw and SB,
We find their respective eigen vectors by,

X1 X2 = eig (inv (S_w)*S_B) ;

Giving us 2 vectors where in X2 is the vector of eigen values and X1 is the vector of eigen vectors for respective values.

Selecting the Top 2 Eigen Value Vectors , we make a W matrix which can be used to find the reduced dimension dataset.
Thus ,

Y= X*W,

Where X is the training data and W is the vector ,which reduces the dimensionality of the problem to K-1, in our case 2.

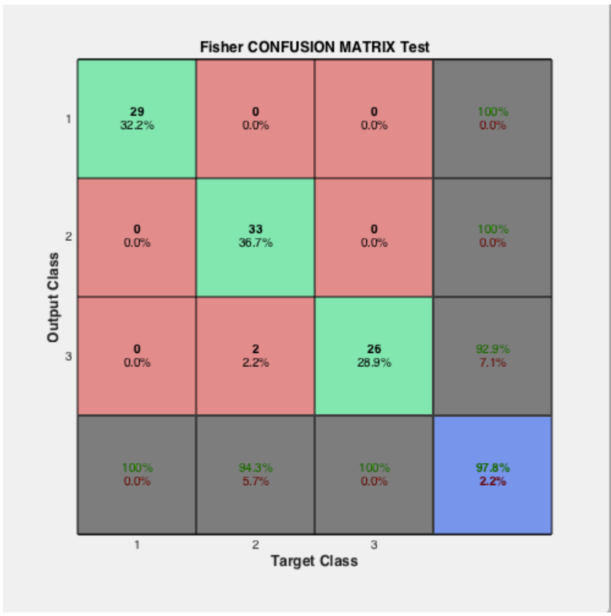
After this we can use LSE method or nearest K nearest neighbors(KNN) to solve the problem of classification. I have used KNN method for classification.

Thus, we have reduced our 13 dimensional data problem into a 2 dimensional data problem with the best informative parameters, as possible. But this can lead to loss of certain information as well, leading to reduction in accuracy.

From Wikepedia : but this is extrmemly apt.

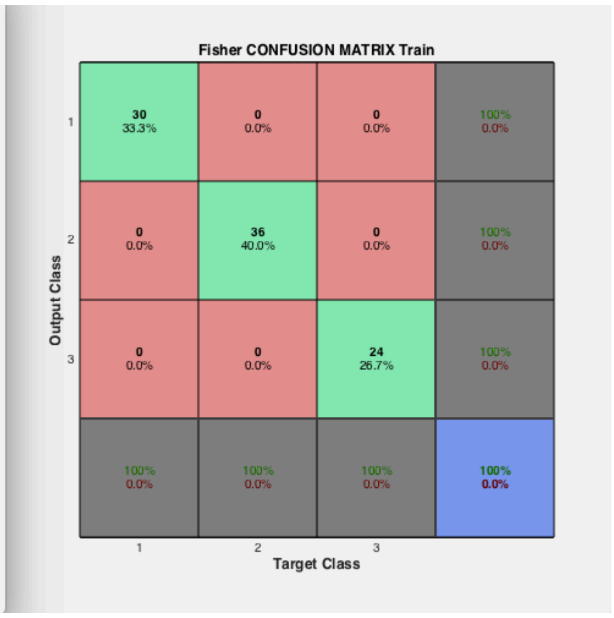
“ LDA is also closely related to principal component analysis (PCA) and factor analysis in that they both look for linear combinations of variables which best explain the data. LDA explicitly attempts to model the difference between the classes of data . ”

Observations:



Fisher CONFUSION MATRIX Test

Output Class \ Target Class	1	2	3	
1	29 32.2%	0 0.0%	0 0.0%	100% 0.0%
2	0 0.0%	33 36.7%	0 0.0%	100% 0.0%
3	0 0.0%	2 2.2%	26 28.9%	92.9% 7.1%
	100% 0.0%	94.3% 5.7%	100% 0.0%	97.8% 2.2%



Fisher CONFUSION MATRIX Train

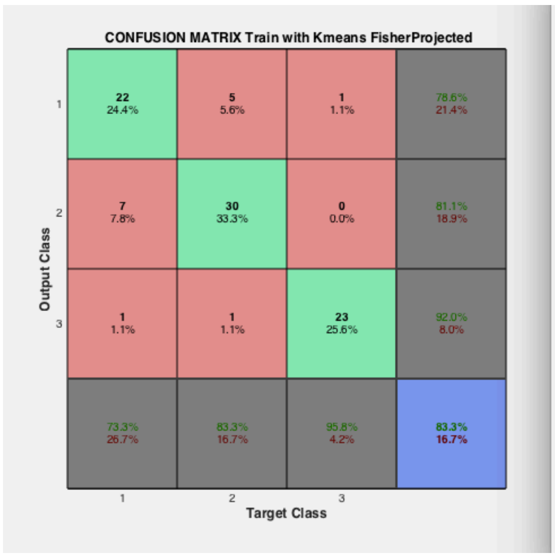
Output Class \ Target Class	1	2	3	
1	30 33.3%	0 0.0%	0 0.0%	100% 0.0%
2	0 0.0%	36 40.0%	0 0.0%	100% 0.0%
3	0 0.0%	0 0.0%	24 26.7%	100% 0.0%
	100% 0.0%	100% 0.0%	100% 0.0%	100% 0.0%

Overfitting:

As we can clearly see this is a case of overfitting. The training data is classified with much more accuracy than the testing data. Thus we can say that reducing dimensionality will get us a better classification.

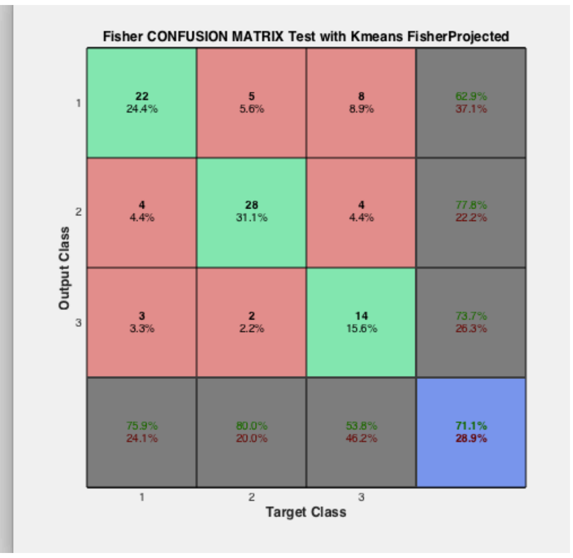
Extra-Credit:

Kmeans on projected data:



CONFUSION MATRIX Train with Kmeans FisherProjected

Output Class \ Target Class	1	2	3	
1	22 24.4%	5 5.6%	1 1.1%	78.6% 21.4%
2	7 7.8%	30 33.3%	0 0.0%	81.1% 18.9%
3	1 1.1%	1 1.1%	23 25.6%	92.0% 8.0%
	73.3% 26.7%	83.3% 16.7%	95.8% 4.2%	83.3% 16.7%



Fisher CONFUSION MATRIX Test with Kmeans FisherProjected

Output Class \ Target Class	1	2	3	
1	22 24.4%	5 5.6%	8 8.9%	62.9% 37.1%
2	4 4.4%	28 31.1%	4 4.4%	77.8% 22.2%
3	3 3.3%	2 2.2%	14 15.6%	73.7% 26.3%
	75.9% 24.1%	80.0% 20.0%	53.8% 46.2%	71.1% 28.9%

Comparision:

Comparing these results with the classification from KNN or Least squares, we can clearly say that it is less accurate. (100% vs 83% for training and 71% vs 73% test). Thus using Kmeans(Unsupervised) doesn't always work because of the underlying assumptions.

Part 3: K-Means Classification

K-means is a clustering method used for classification. This is unsupervised learning. Thus we do not provide the training targets before hand. We train it by finding the centers of the respective centroids and then calculating the Euclidean distance to update the cluster centers for best possible clusters.
There are certain assumptions made before we run the K-means Algorithm :

a. The radius of each cluster is almost the same.
b. You know how many clusters you want. (In our case we have 3 because of the three classes.)
c. The clusters are roughly spherical.
d. The clusters contain the same number of points.

I followed the same algorithm explained by Prof. Collins in class.

1. Select the number of clusters you want.
2. Assign a centroid for each dimension. Thus in our case I assigned a centroid for each dimension and for each class randomly.(Explained in the code).
3. Make a check_diff variable which will check when the distance of the calculated centroid and the one that's old one is 0. When its 0 we terminate the loop for recalculating the centroid.
4. I calculated the Euclidean distance and then assigned the points to each cluster.
5. Recalculate the centroids.
6. Check the difference between centroid and old centroid and if it is 0, end loop or else go back to step 3.

Equation for Classification:

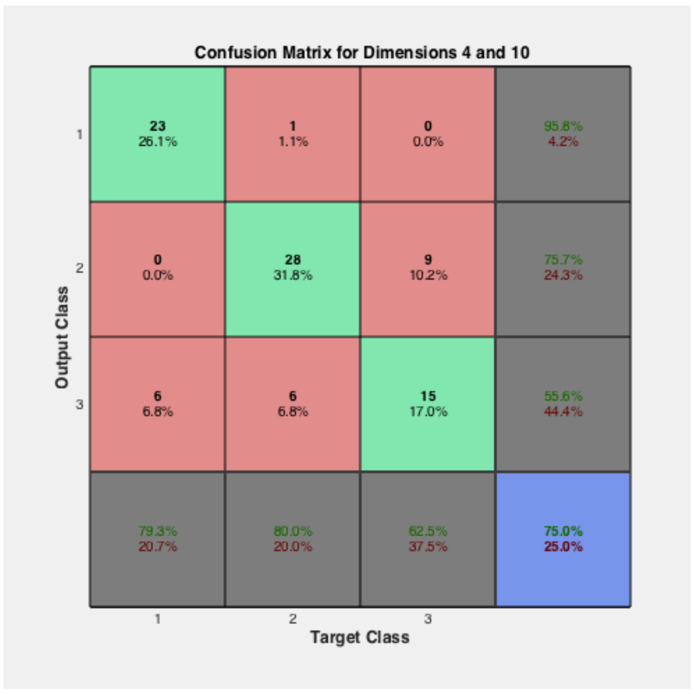
$$\sum_{n=1}^{N}\sum_{k=1}^{K} z_{nk}\,\|x_n - \mu_j\|^2$$

(taken from the slides of Prof. Collins)
here , Znk= Label of each Class.
Xn= input point for that specific dimension
Uj= centroid for that dimension

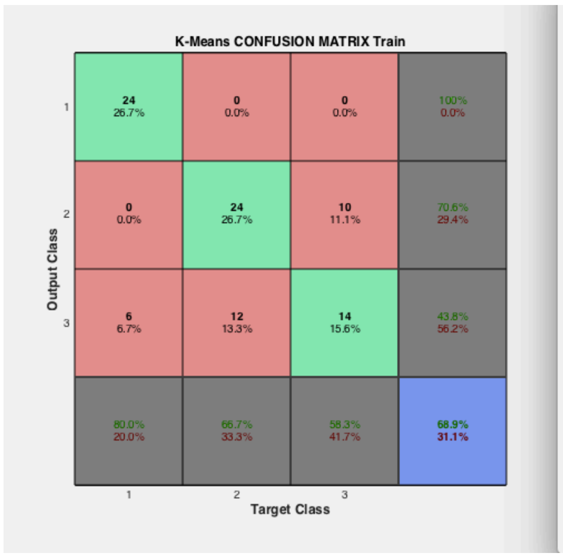
Equation for Recalculating Centroid :

$$\mu_j = \frac{\sum_{n=1}^{N} z_{nj}\, x_n}{\sum_{n=1}^{N} z_{nj}}$$

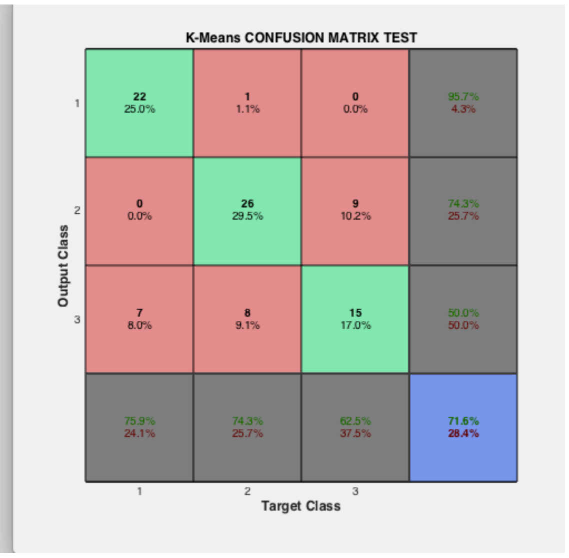
Observations :


Confusion Matrix for Dimensions 4 and 10
Output Class
Target Class

	1	2	3	
1	23 26.1%	1 1.1%	0 0.0%	95.8% 4.2%
2	0 0.0%	28 31.8%	9 10.2%	75.7% 24.3%
3	6 6.8%	6 6.8%	15 17.0%	55.6% 44.4%
	79.3% 20.7%	80.0% 20.0%	62.5% 37.5%	75.0% 25.0%

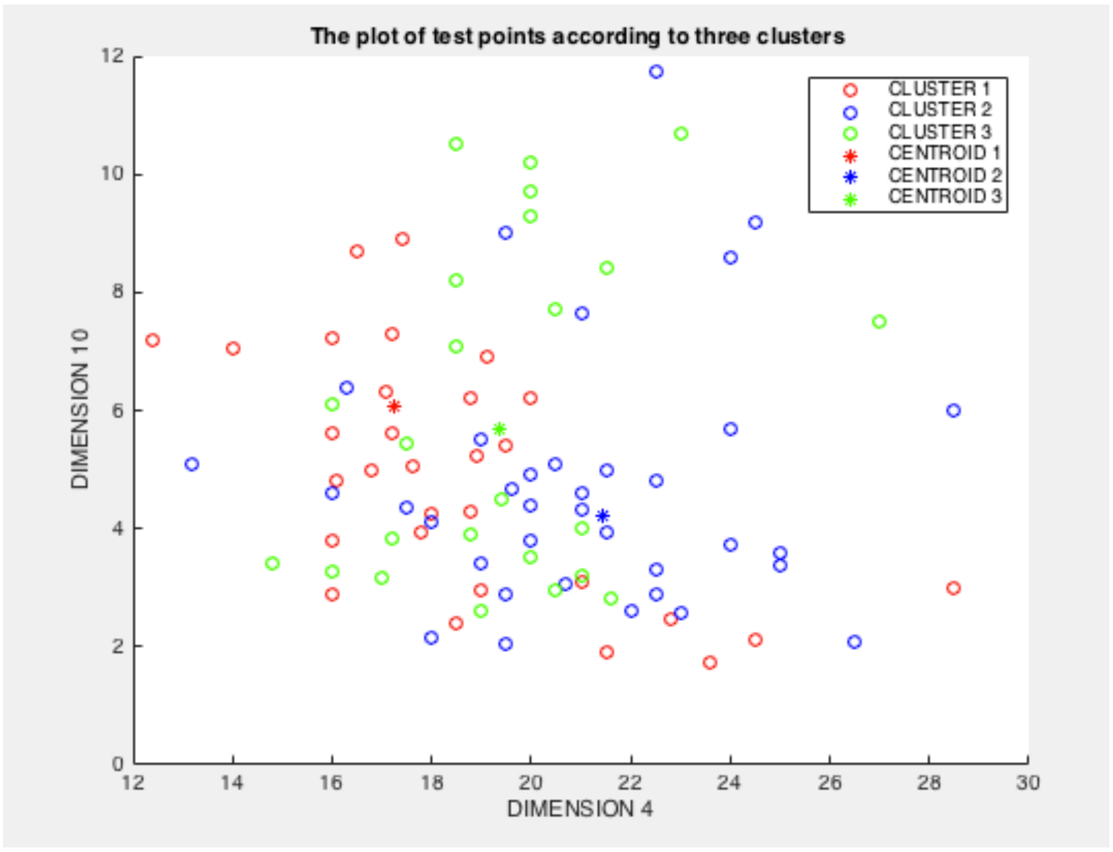

K-Means CONFUSION MATRIX Train
Output Class
Target Class

	1	2	3	
1	24 26.7%	0 0.0%	0 0.0%	100% 0.0%
2	0 0.0%	24 26.7%	10 11.1%	70.6% 29.4%
3	6 6.7%	12 13.3%	14 15.6%	43.8% 56.2%
	80.0% 20.0%	66.7% 33.3%	58.3% 41.7%	68.9% 31.1%


K-Means CONFUSION MATRIX TEST
Output Class
Target Class

	1	2	3	
1	22 25.0%	1 1.1%	0 0.0%	95.7% 4.3%
2	0 0.0%	26 29.5%	9 10.2%	74.3% 25.7%
3	7 8.0%	8 9.1%	15 17.0%	50.0% 50.0%
	75.9% 24.1%	74.3% 25.7%	62.5% 37.5%	71.6% 28.4%

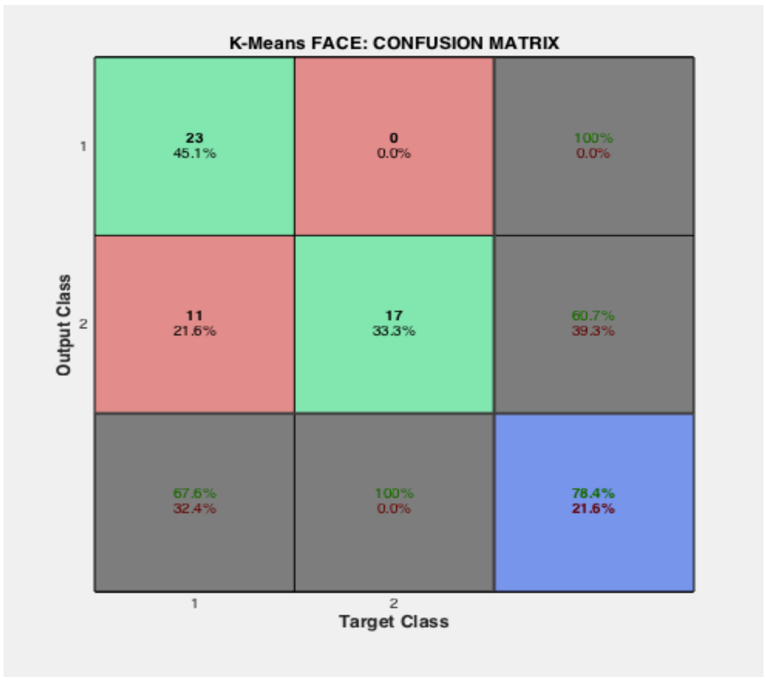
If we check the variable (iteration) in the code we can clearly see that it takes more than one iteration for getting the accurate centroid. However even after that we can be certain that we cannot completely classify all the given test data. Which in turn means that we don't get completely separate clusters as such. We can see that in the next Figure.



From a statistical viewpoint, the clusters obtained by k-mean can be interpreted as the Maximum Likelihood Estimates (MLE) for the cluster means if we assume that each cluster comes from a spherical Normal distribution with different means but identical variance (and zero covariance).

This touches upon a general disadvantage of the k-means algorithm: k-means works best for images with clusters that are spherical and that have the same variance. (as were the assumptions) But our data WINE wasn't a completely clustered data. Hence the reduced accuracy.

Observations for Face Data :



OverFitting:

K-Means is relatively an efficient method. However, we need to specify the number of clusters, in advance and the final results are sensitive to initialization and often terminates at a local optimum. Here we have three clusters as we know there are three classes, but in general, a large *k* probably decreases the error but increases the risk of overfitting. Here we see some overfitting when comparing testing and training data.

PLOTFUNCTION:

I have used the PLOTCONFUSION function on the vectors that I created stating the class of each input. The vectors have 3/2 colums representing the classes , one of which will be one only when the data sample belongs to that specific class. We can use this PLOTCONFUSION function as we are given the final target outputs and , when we calculate the Y, we get the calculated outputs. The dimensionality is considered while calculating the outputs themselves, by the W vector. I have also plotted the clustering diagram taking into consideration 2 dimensions(4 and 10).